PROVIDING REMOTE PROCESSING SERVICES OVER A DISTRIBUTED COMMUNICATIONS NETWORK

Technical Field

This application relates in general to a method, apparatus, and article of manufacture for providing a set of computing services to remotely executing client processes that execute upon a server connected to the client process over a distributed communications network, and more particularly to a method, apparatus, and article of manufacture for providing web-based processing services to remote users communicating with a server over the Internet.

Background

The growth of the Internet and related network communication networks has given rise to an increasingly large numbers of distributed information processing systems in which individual users obtain information from an ever increasing number of sources. Currently, web servers typically provide a mechanism to permit content data to be displayed within a web browser running on a remote client. Over time, this content data has evolved to include static pages, dynamically created pages, and pages that include programmable functionality that executes on the server to generate content displayed to a user within the framework of the browser. This data exchange model typically uses a web browser that transfers a "page" of data that is to be displayed to the user in some manner.

Web servers, however, have not provided remote execution of processing functions that may reside on a server. All of the processing functions that are typically performed by a web server relate to processing database queries and similar data retrieval operations upon data that is resident on the server. Web servers do not typically receive a block of input data, process a function upon the data,

and return a resultant set of data. In addition, web servers do not typically operate upon input data sets to generate output data sets as if the web server was responding to a remote process or function call. In spite of the fact that significant computational capacity is resident within web servers, these servers typically provide a "read-only" type of data retrieval function that are based upon user-generated requests.

If web servers provide an interface that allows remote client processes to execute functions resident on the servers using the web as a communications mechanism, web servers provide the equivalent functionality of a remote procedure call for users seeking access to processing functions resident on the server. Such an interface would change the Internet from a means to share and display content data to a distributed processing environment in which web servers provided processing services to remote clients.

Summary

A method, apparatus and article of manufacture for creating Internet web servers that provide processing services, in addition to data and visual content, to remote clients with access to servers. The client processes communicate with these service providing servers over a distributed network like the Internet using standard HTTP communications protocol and similar data exchange languages. Client processes send an HTTP request to a remote server for processing. This processing request may contain input data that is to be used in responding to the request. The server processes the request using the input data, and possibly other data obtained from remote databases, and returns a resultant specified data packet. This processing request, may be initiated using a web browser from an HTML based web page or using a smart client process that simply sends a

processing request along with input data and consumes the resultant data packet. The server provides these web services accessed by the client process by allowing the client to access a URL referencing a source code file containing class specifications that may be dynamically compiled into an executable object whenever an executable object corresponding to the current version of the source code referenced by a URL does not exist on the server.

One aspect of the present invention is a method and an article of manufacture containing data readable by a computing system and encoding instructions for providing remote computing services executed upon a server having mass data storage and network communications to remote clients. The method stores source code file within the mass storage of the server, receives a processing service request from a remote client corresponding to a requested processing service specified within the source code file, activates a data processing object corresponding to the processing service requested from the remote client to generate a data response corresponding to the processing service request, formats the data response into a text based data response packet, and transmits the text passed data response packet to the remote client in response to the processing service request. The data processing object corresponds to a compiled data processing class for the requested processing service contained within the source code file.

Another aspect of the present invention is a network server for providing remote computing services executed upon the network server having mass data storage for storing a source code file containing a data processing class corresponding to a data processing object that implements a requested remote processing service. The network server has a network interface I/O module for receiving a processing service request from a remote client corresponding to a requested processing service specified within a source code file and for transmitting the text based data response packet to

the remote client in response to the processing service request, a request processing module for generating a data response corresponding to the processing service request, and the request processing module activates the data processing object corresponding to the requested processing service and formatting the data response into a text based data response packet. The data processing object corresponds to a compiled data processing class for the requested processing service contained within the source code file.

Yet another aspect of the present invention is a method and an article of manufacture containing data readable by a computing system and encoding instructions for automatically compiling server executed source code corresponding to data processing objects used to provide remote processing services upon receipt of a request for a remote processing services. The method stores a source code file within the mass storage of the server, receives a processing service request from a remote client corresponding to a requested processing service specified within the source code file, extracts from a packet payload body data block an ID corresponding to the requested processing service, determines whether the data processing object corresponding to the requested processing service is located within a web services library, and determines whether any data processing object found within the web services library correspond to the data processing object generated by compiling the data processing class for the requested processing service contained within the source code file. If the data processing object does not corresponds to the data processing object generated by compiling the data processing class for the requested processing service contained within the source code file, the method automatically compiles the source code file to generate the data processing object for the requested processing service.

Yet another aspect of the present invention is a method and an article of manufacture containing data readable by a computing system and encoding instructions for automatically creating data exchange schema data on a network server corresponding to remote processing services provided by the network server for source code corresponding to data processing objects used to provide the remote processing services upon receipt of a request from a client process. The method storing a source code file within the mass storage of the server, compiling the source code file to generate a data processing object, and automatically generating the data exchange schema data for

the data processing object generated when the source code file is compiled to generate the data processing object that provides the requested processing service.

These and various other features as well as advantages, which characterize the present invention, will be apparent from a reading of the following detailed description and a review of the associated drawings.

Brief Description of the Drawings

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

Fig. 1a illustrates a network based processing system providing processing services to remote clients according to one embodiment of the present invention.

Fig. 1b illustrates a legal word processing application using a network based processing system providing processing services to remote clients according to one embodiment of the present invention.

Fig. 1c illustrates a sales tax calculating application using a network based processing system providing processing services to remote clients according to one embodiment of the present invention.

Fig. 2 illustrates a general purpose computing system for use in implementing as one or more computing embodiments of the present invention.

Fig. 3 illustrates an Internet based processing system providing processing services to remote users according to another embodiment of the present invention.

Fig. 4 illustrates a network server providing processing services to remote users according to yet another embodiment of the present invention.

Fig. 5 illustrates an HTTP packet used by a network server providing processing services to remote users according to an embodiment of the present invention.

Fig. 6 illustrates an XML Request Packet Payload Body and corresponding XML Response Packet according to another embodiment of the present invention.

Fig. 7 illustrates a network server providing processing services to remote users according to yet another embodiment of the present invention.

Fig. 8 illustrates a compiler module within a network server providing processing services to remote us according to an embodiment of the present invention.

Fig. 9 illustrates a plurality of output data types generated by a compiler module according to an embodiment of the present invention.

Fig. 10 illustrates an operational flow for the server-based processing system according to an embodiment of the present invention.

Fig. 11 illustrates an operational flow for a compiler module according to an embodiment of the present invention.

Fig. 12 illustrates an operational flow for a programming method for providing processing services to remote users according to one embodiment of the present invention.

Detailed Description

This application relates in general to a method, apparatus, and article of manufacture for providing a network based processing system providing processing services to remote clients.

Fig1a illustrates a network based processing system providing processing services to remote clients according to one embodiment of the present invention. Remote users use client processors

121-124 to communicate over a communications network like the Internet 101 to communicate to one or more server processors 102 to obtain data and processing services. Web servers typically have provided a mechanism to transmit content data to be displayed within a web browser running on a remote client. Over time, this content data has evolved to include static pages, dynamically created pages, and pages that include programmable functionality that executes within the framework of the browser. Web servers have evolved to utilize and support this functional evolution of web browsers. This data exchange model in which a user interface is projected across the web using a web browser to transfers a "page" of data to be displayed to the user

Remote servers 102, may provide remote execution of processing functions that may reside on a server according to one embodiment of the present invention. In addition to of processing database queries and similar data retrieval operations upon data that is resident on the server, servers typically receive a block of input data, process a function upon the data, and return a resultant set of data. Web servers 102 provide an interface that allows remote client processes to execute functions resident on the servers using the web as a communications mechanism, these servers can provide the equivalent functionality to a remote procedure call to processing functions resident on the server using currently used Internet communications mechanisms. These server resident functions may provide various computing functions to users of the remote clients as discussed in reference to Figures 1b-1c.

Fig. 1b illustrates a legal word processing application using a network based processing system providing processing services to remote clients according to one embodiment of the present invention. A word processing application displays a typical data window 130 to a user on a client processor 121. The word processing window 130 contains a menu bar 131 having a plurality of pull-

down menus, including a "Citations" 132 menu that is related to legal case citations. A user enters the text for a document 134 into the application 130 as one uses any word processor. When a citation to a court decision 133 is encountered, the word processor can display relevant information about the decision if this data is available in a separate window 140.

Currently, word processors do not provide such a feature because the required data related to these decisions, or any other relevant data, is typically so voluminous that it is not typically present upon the client processor. The web services of the present invention provides a mechanism to provide this data from a web based remote processor for easy importation into the word processing application. Databases containing this case decision data are currently present on the Internet. What is needed is a processing mechanism to connect the client applications, such as word processing applications with these remotely located databases.

Using web services of the present invention, the word processor would be told, or recognize the citation 134 as such a data item and search the Internet for a data source for the relevant data. The word processor could possess preferences stating where the application should search for the data and the relevant format for the information to be displayed. When the user activated a link associated with the citation, the case data window 140 would be displayed. The case data window 140 may contain a full citation to the case 141, a brief summary of the case 142, and one or more links to other data such as a full text version of the opinion 143 and a listing of decisions citing this decision such as the listing provided to lawyers by Shepard's Citations 144. Any relevant data that can be retrieved from a remote database could be retrieved and displayed using this process. This mechanism would permit this data to be available to users without having an entire copy of the relevant database to be resident on the client computer 121. The word processor 130 can also be

configured to store some or all of this relevant data within the document for presentation upon a client processor 121 that is not always connected to the Internet or may merely store links to the data for retrieval when requested by a user. These options may easily be configurable by a user using preference settings within the word processor.

Fig. 1c illustrates a sales tax calculating application using a network based processing system providing processing services to remote clients according to one embodiment of the present invention. Rather than providing a richer data content environment to users as discussed above, the web services may also be used to provide a processing function that uses a remote database that a user does not wish to possess or maintain. One such example processing function is a sales tax calculator. The applicable amount of sales tax to be applied to a given sale varies greatly depending upon the jurisdiction to whom the tax is to be paid.

For example, 50 US states each possess a different state sales tax amount that varies from state to state. Delaware does not collect a sales tax where Minnesota does. Within a given state, such a Minnesota 152, an individual county within Minnesota also may apply a county wide sales tax. Within one county, such as Hennepin County 154, various local cities and towns, such as Minneapolis 156, may also apply local sales tax. For all of these tax rates, the applicable rate may be different depending upon the goods and services 158 being sold.

Merchants that sell goods and services to multiple jurisdictions find the calculation of the correct rate difficult as the number of relevant jurisdictions increases. Rather than have the merchants be responsible for determining the correct rate by address/location and type of goods, a web service can accept a buyer's address, information regarding the type of goods purchased, and a

purchase amount for the goods and then calculate and return a tax amount. This web service would maintain a single database of the applicable rates and update the rates as individual state, county, and local jurisdictions modify the applicable tax rate. The merchant would simply access the web service, providing a request that provides an address or zip code 161, a goods type description, 162, and a price 163 in a single request 160. The web service uses the request data 160 to look up the applicable tax rate based upon the address and type of goods and applies the rate to the price to determine a tax amount. This tax amount is returned to the merchant in a return data record 170.

Any processing can be implemented using the above processing models. The complexity of the processing performed by the web service could be limitless, depending upon the processing power of the servers, the data to be processed, and the amount of time a client will reasonably wait for a response. Similarly, any database that provides data to a processing system, and any number of different databases, can be used in such a system. The server and client simply need to express the request data format, the response data format, and the processing to be performed in a same manner.

With reference to Figure 2, an exemplary system for implementing the each of the computing systems, including both web servers and remote clients, include a general-purpose computing device in the form of a conventional personal computer 200, including a processor unit 202, a system memory 204, and a system bus 206 that couples various system components including the system memory 204 to the processor unit 200. The system bus 206 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 208 and random access memory (RAM) 210. A basic input/output system 212 (BIOS), which contains basic routines that help transfer information between elements within the personal computer 200, is stored

in ROM 208.

The personal computer 200 further includes a hard disk drive 212 for reading from and writing to a hard disk, a magnetic disk drive 214 for reading from or writing to a removable magnetic disk 216, and an optical disk drive 218 for reading from or writing to a removable optical disk 219 such as a CD ROM, DVD, or other optical media. The hard disk drive 212, magnetic disk drive 214, and optical disk drive 218 are connected to the system bus 206 by a hard disk drive interface 220, a magnetic disk drive interface 222, and an optical drive interface 224, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, programs, and other data for the personal computer 200.

Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 216, and a removable optical disk 219, other types of computer-readable media capable of storing data can be used in the exemplary system. Examples of these other types of computer-readable mediums that can be used in the exemplary operating environment include magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), and read only memories (ROMs).

A number of program modules may be stored on the hard disk, magnetic disk 216, optical disk 219, ROM 208 or RAM 210, including an operating system 226, one or more application programs 228, other program modules 230, and program data 232. A user may enter commands and information into the personal computer 200 through input devices such as a keyboard 234 and mouse 236 or other pointing device. Examples of other input devices may include a microphone, joystick, game pad, satellite dish, and scanner. These and other input devices are often connected to the

processing unit 202 through a serial port interface 240 that is coupled to the system bus 206. Nevertheless, these input devices also may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB). A monitor 242 or other type of display device is also connected to the system bus 206 via an interface, such as a video adapter 244. In addition to the monitor 242, personal computers typically include other peripheral output devices (not shown), such as speakers and printers.

The personal computer 200 operates in a networked environment using logical connections to one or more remote computers, such as a web server 246. The remote computer 246 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the personal computer 200. The network connections include a local area network (LAN) 248 and a wide area network (WAN) 250. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, the personal computer 200 is connected to the local network 248 through a network interface or adapter 252. When used in a WAN networking environment, the personal computer 200 typically includes a modem 254 or other means for establishing communications over the wide area network 250, such as the Internet. The modem 254, which may be internal or external, is connected to the system bus 206 via the serial port interface 240. In a networked environment, program modules depicted relative to the personal computer 200, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary, and other means of establishing a communications link between the computers may be used.

Additionally, the embodiments described herein are implemented as logical operations performed by a computer. The logical operations of these various embodiments of the present invention are implemented (1) as a sequence of computer implemented steps or program modules running on a computing system and/or (2) as interconnected machine modules or hardware logic within the computing system. The implementation is a matter of choice dependent on the performance requirements of the computing system implementing the invention. Accordingly, the logical operations making up the embodiments of the invention described herein can be variously referred to as operations, steps, or modules.

Fig. 3 illustrates an Internet based processing system providing processing services to remote users according to another embodiment of the present invention. A remote client 122 communicates to a server 102 over a communications network 101 to transmit a web service request and receive a corresponding response. In one embodiment, this communication is performed using the well known HTTP communications protocol; although, any request/response communications protocol could be used without deviating from the spirit and scope of the present invention as recited in the attached claims.

Within the server 102, a web server I/O module 301 provides the communication processing necessary to communicate with the remote client 122. The web server I/O module 301 receives the incoming request, extracts the relevant data from the request and passes the request to a Web services processing module 302, and returns the response generated by the web services processing module 302 to the client 122. This module 301 is responsible for conforming the data to the relevant communications protocol.

The web services processing module 302 accepts the requests from the I/O module 301 and generates a response. This processing module 302 uses processing modules contained within a web services library 303 to perform the web services processing request. This processing module 302 also retrieves any data needed to perform the processing request from one or more databases 115 using a database query interface module 304. This interface module 304 performs the communication functions between the server 102 and the database 115. This interface module 304 also performs any data formatting functions necessary to present the needed data to the processing module 302 in a usable format. Alternatively, a web services library may directly access a database and return the results directly to processing module 302. Once the processing module 302 has generated the response corresponding to the requested web service, the response is sent back to the client 122 using the web interface I/O module 301.

Fig. 4 illustrates a network server providing processing services to remote users according to yet another embodiment of the present invention. The server 102 consists of the web interface I/O module 301 contains a TCP/IP communications module 311 and a server securitymodule 312 to perform its web communications function. The TCP/IP communications module 311 receives TCP/IP communications data packets and determines which packets correspond to web server requests. These web server requests are sent to a Server security module 312 for processing.

The Server security module 312 receives these request packets that typically contain a URL that identifies the provides an indication of the identity of the request, and the data corresponding to the request that is to be processed. If the Server security module 312 determines that the URL corresponds to a web service request, the request is sent to the web services processing module 302 for handling. The Server security module 312 may also receive other types of requests such as

Active Server Page (ASP), Common Gateway Interface (CGI) script and other web server requests that are handled by other functions not part of the present invention.

The web services processing module 302 comprises of a request module 321, a compiler module 322, and a data acquisition module 323. The request module 321 receives the incoming web services request and determines whether the requested web service exists on the server 102 and if it is contained within the web services library for use in generating the response to the incoming request. Because the request is provided in the form of a URL, the request typically corresponds to a file that contains instructions providing the processing to be performed. For example, URL, such as http://www.microsoft.com/example/index.html, may provide a reference to a file that contains an HTML specification for a web page. This file may contain human readable source code in any programming language such as HTML, C++, or any other programming language. This file may contain any number of processing modules and exposed web services that may be accessed by a client 122.

The request module 321 determines whether this file has been previously compiled and stored within the web services library 303. If the file has been compiled, and if the compiled version corresponds to the current version of the URL file, the processing object module stored in the library is retrieved for use in servicing the incoming request. If the URL file has not been compiled, or if the URL has been modified since the file was compiled, a compiler module 322 is used to compile the URL file into a executable processing object module. The compiler module 322 compiles the current version of the file referenced by the URL, stores the newly compiled object within the library 303 for later use, and passes the compiled object to the request module 321 for processing.

Once the request module 321 receives the executable processing object module, the request module 321 services the incoming request by executing the executable processing module using any input data provided with the incoming request. If the executable processing module requests data from a database 115 as part of its operation, the request module sends a data request to the data acquisition module 323. This data module 323 obtains the needed data using the database query interface module 304 as discussed above and returns the data to the request module 321 for use by the executable processing object.

Fig. 5 illustrates an HTTP packet used by a network server providing processing services to remote users according to an embodiment of the present invention. The HTTP Packet 500 contains a HTTP header 501 to perform the standard HTTP communications function. HTTP header 501 typical contains information relating to the URL of the request, information regarding the identity and type of requesting client, and return address information associated with the client for sending a response to the request.

HTTP Packet 500 also contains Packet Body Payload 502 that contains information to be used in the request. In cases where a web service is requested by the packet, the packet body payload 502 contains a description of the web service request and any data to be processed as part of the web services request. In an exemplary embodiment, this payload contains a textual description of the request and data in an XML format using the XML language , as specified in the Extensible Markup Language (XML) v1.0 specification as defined by the w3c.org standards organization. In particular, the present invention uses SOAP standard specification for XML specification of the data specification protocol. A complete description of the SOAP standard can be found in the Simple Object Access Protocol (SOAP) v. 1.1 specification as defined by the w3c.org standards

organization. Of course, any particular XML standard, or any similar data exchange specification language including a Web Services Description Language (WSDL), may be used without deviating from the spirit and scope of the present invention as recited in the attached claims.

Fig. 6 illustrates an exemplary XML Request Packet Payload Body and corresponding XML Response Packet according to another embodiment of the present invention. An XML payload body contains a reference to the name of the web service to be performed 602 as well as one or more input data arguments 603 that are to be passed to the web service for processing. The payload also contains the necessary references to identify the specific data specification standard as well as any other labels needed to conform the input data to the specific data specification standard. Of course, one skilled in the art will recognize that this particular embodiment of the present invention that utilizes the XML, SOAP, and WSDL standards may possess alternate embodiments that utilize a data payload body 601 to transmit data between a web server and a remote client without deviating from the spirit and scope of the present invention as recited in the attached claims. The XML response 611 generated by the incoming web service request contained in the XML payload body 601 also contains data to be returned 612 and its values 613 as well as the necessary references to identify the specific data specification standard as well as any other labels needed to conform the input data to the specific data specification standard.

Fig. 7 illustrates a network server providing processing services to remote users according to yet another embodiment of the present invention. The request module 321 contains a series of processing modules that process a web service request contained in a HTTP packet payload body. The packet payload body is first read by an Obtain XML payload body module 701. The module 701 extracts the relevant payload body data from the incoming packet for use in the processing. An

Examine Web Method ID module 702 examines the XML data from the payload data to determine the identity of the web services method to be performed. The examine module 702 determines if the method can be found. The examine module 702 will also determine if a compiled version of a processing object exists within the web services library 303. If the examine module 702 finds the processing object, it is requested from the compiler module 322. If the stored version of the processing object does not correspond to the current version of the object, the compiler module 322 will compile the object before returning the object to be used to service the request.

Once an executable object is obtained, a deserialize input argument module 703 reads and marshals the input arguments 603 from the payload body 502 into data that can be passed to the executable object obtained from the compiler module 322. The input arguments and the executable object are passed to an invoke web method module 704 to execute the object using the input data to generate a response to the web services request. If data from a database 115 is needed by the executable object, the data request is sent to a data acquisition module 323 for servicing through an interface module 304.

The executable object completes its processing and generates response data that is to be returned to the requesting client 122. The response data is converted into XML textual data in the appropriate format in a Serialize Return Value module 705. This module 705 accepts data from the executable object and converts the data into a textual based data stream that is sent back to the client 122 as part of the HTTP response to the incoming request using the Return HTTP Response Module 706.

Fig. 8 illustrates a compiler module within a network server providing processing services to remote us according to an embodiment of the present invention. As discussed above in Fig, 7, the Examine Web Method ID module 702 uses the incoming URL to identify the web services method corresponding to the incoming request. As also stated above, the URL references a text file containing source code for the module or modules needed to provide the web service. These modules may consist of any number of processing modules in any programming language. The modules that are exposed as accessible by client processors 122 are identified as publicly available methods. The file may also contain other modules, if needed, that are only accessible within other modules. With such an architecture, a web service may be constructed using any number of libraries of executable modules containing a set of entry points accessible by clients without exposing all of the modules within these libraries if that is desired.

The examine module 702 checks the URL referenced file to determine if the requested web service method is contained in the referenced file. If the method is present, the examine module sends a request to the compiler module 322 to obtain the executable object. Within the compiler module 322, a obtain/parse web method module 801 receives the packet payload body data and determines if a compiled version of the web service executable object is stored in the web services library 303. If an executable object is found, the module also checks to see if the stored version corresponds to the current version of the URL file. If the stored object does match the URL referenced file, the object is simply obtained from the library 303 and returned to the examine module 702.

If the obtain/parse module 801 determines that the library 303 does not contain a usable executable object, the URL file needs to be compiled into an executable object. This compilation

operation will only occur the first time the URL referenced file is accessed following is storage onto the server 102. All other requests will use the library version until the file is modified.

If the URL referenced file needs to be compiled, the URL source code file is read and compiled by a compiler corresponding to the programming language specified to be used in the source file by the compile class module 802. This compiler module 802 will compile all objects in the source code file. These objects may use any other executable objects present on the server. As such, these objects may reference other objects contained in the source code file as well as objects present in other source code filed stored on the server that can be accessed using a URL reference to a source code file. The same process is followed when the these external objects are referenced to determine if the objects exist within the source code files referenced by the URL and to determine whether the library 303 contains usable executable objects that do not require compilation.

Once the compilation has been completed, a web method attribute module 803 identifies all publicly available attributes to publicly accessible web service methods. These methods and corresponding attributes are the values that may be passed to the server from the client as part of a web service request. A cache compiled object module 804 stores the newly compiled object, along with a description of its publicly accessible methods/attributes into the web services library 303 for use in subsequent requests for this service before the executable object is returned to the request module 321.

Fig. 9 illustrates a plurality of output data types generated by a compiler module according to an embodiment of the present invention. The above discussion of the compiler module presumes that a compiler will generate an executable object from a source code file. This compiler 322, and

all of its modules 801-804, may generate other forms of data in addition to executable objects. Specifically, the compiler module 322 generates a compiled class module 901 containing one or more types of data. These types of data include an executable class module 910 as discussed above. In addition, the compiled class module 901 may include a description of the publicly exposed web services contained within the source code file referenced by the URL. This description may include a description of the function or functions performed by each of the publicly exposed web services. As part of these descriptions, a definition for the input arguments, and any publicly accessible attributes, as well as a definition of the output data generated by the web service are provided. This description may be presented in any number of formats.

One such format for the description is an Browser Displayable HTML Description. The compiler will generate an HTML page description 920 that contains a description of the functions performed in the exposed web services. These descriptions are contained within data found in the source code files. In addition, a description of the input and output data arguments 921-922 for the each web service will be provided. This data argument descriptions 921-922 may also present fields 923 permitting a user to enter values for the input arguments using a web browser. The return value description for each web service contains a invoke button 924 that causes the web service to be called using the input argument values contained in the web page fields 923. The web browser will display the returned XML result data from the web service request.

A user and developer may use this HTML description to determine the input and output arguments for a web service as well as test a web service interactively while it is being debugged. The server 102, within its ISS server module 312 can determine if the incoming request is coming

from a web browser or from some other client process using HTTP header data to determine if the HTML version is to be returned.

In addition to the HTML description 920, a textual schema description in a data exchange schema specification format may be provided. XML as currently specified by the w3c.org standards organization, provides mechanisms for specifying how schema descriptions for data being exchanged using XML are to be specified. In an exemplary embodiment, this schema description is specified using a WDSL when WDSL is used to specify the XML data within the packet payload body data. Other schema description languages, such as RDF as proposed by the w3c.org standards organization, may also be used without deviating from the spirit and scope of the present invention as recited within the attached claims. Users may obtain these schema descriptions to determine the format and functions of the web services and corresponding input and output arguments.

Fig. 10 illustrates an operational flow for the server-based processing system according to an embodiment of the present invention. This operational flow corresponds to the server based processing as discussed above in reference to Fig. 7. The operational flow starts 1001 and packet payload body is first read by an Obtain XML payload body module 1011. The obtain data module 1011 extracts the relevant payload body data from the incoming packet for use in the processing. An Examine Web Method ID module 1012 examines the XML data from the payload data to determine the identity of the web services method to be performed. A test module 1013 determines if the method can be found, and if present determines if a compiled version of a processing object exists within the web services library 303. If the test module 1013 finds the processing object needs to be compiled, a compile web method module 1014 compiles the object to generate an executable object to be used to service the request.

Once an executable object is obtained, a deserialize input argument module 1015 reads and marshals the input arguments 603 from the payload body 502 into data that can be passed to the executable object. The input arguments and the executable object are passed to an invoke web method module 1016 to execute the object using the input data to generate a response to the web services request.

The executable object completes its processing and generates response data that is to be returned to the requesting client 122. The response data is converted into XML textual data in the appropriate format in a Serialize Return Value module 1017. This module 1017 accepts data from the executable object and converts the data into a textual based data stream that is sent back to the client 122 as part of the HTTP response to the incoming request using the Return HTTP Response Module 1018 before the processing ends 1002.

Fig. 11 illustrates an operational flow for a compiler module according to an embodiment of the present invention. The compiler module processing starts 1101 and an obtain/parse web method module 1111 receives the packet payload body data and parses the data to identify the requested web service. Test module 1115 determines if a compiled version of the web service executable object is stored in the web services library 303, and if found, the module 1115 also checks to see if the stored version corresponds to the current version of the URL file. If the stored object does match the URL file, no compilation is needed. The processing proceeds to an obtain library module 1116 where the object is simply obtained from the library 303 and returned before the processing ends 1102.

If the test module 1115 determines that the library 303 does not contain a usable executable object, the URL file needs to be compiled into an executable object. A compile class module 1112

will compile all objects in the source code file. These objects may use any other executable objects present on the server. As such, these objects may reference other objects contained in the source code file as well as objects present in other source code filed stored on the server that can be accessed using a URL reference to a source code file. The same process is followed when the these external objects are referenced to determine if the objects exist within the source code files referenced by the URL and to determine whether the library 303 contains usable executable objects that do not require compilation.

Once the compilation has been completed, a web method attribute module 1113 identifies all publicly available attributes to publicly accessible web service methods. These methods and corresponding attributes are the values that may be passed to the server from the client as part of a web service request. A cache compiled object module 1114 stores the newly compiled object, along with a description of its publicly accessible methods/attributes into the web services library 303 for use in subsequent requests for this service before the processing ends 1102.

Fig. 12 illustrates an operational flow for a programming method for providing processing services to remote users according to one embodiment of the present invention. The processing begins 1201 and a source code file is created in a desired programming language by a Create source code module 1211. The created source code file is saved onto a server 102 using a save source code module 1212. The storage location of the source code file corresponds to the URL for the web service request for any exposed web service contained within the source code file.

Once the source code file is saved, the processing continues once a web service request is received by a receive processing request module 1213 in the server 102. The receive request module

1213 receives the request for the URL corresponding to the source code file that contains a payload body referencing the exposed web service. A test module 1214 determines if the requested web method needs to be compiled. This test compares any compiled version in the web service library 303 with the source code file to see if the versions correspond to each other. If the test module 1214 determines that no compilation is necessary, an Obtain Cached Object Class module 1215 retrieves the executable object from the library 303. A Process request module 1219 activates the retrieved object and passes any input arguments parsed from the payload body data to the object. The executable object generates output data which is formatted by the process module 1219 into an HTTP response sent to the requesting client.

If test module 1214 determines that the source code file needs to be compiled, a compile class module 1215 will compile all objects in the source code file. Once the compilation has been completed, a web method attribute module 1217 identifies all publicly available attributes to publicly accessible web service methods. These methods and corresponding attributes are the values that may be passed to the server from the client as part of a web service request. A cache compiled object module 1218 stores the newly compiled object, along with a description of its publicly accessible methods/attributes into the web services library 303 for use in subsequent requests for this service before the object is passed to the process module 1219. The process module 1219 again activates the object to generate a response and the processing ends 1102.

Figure 2 illustrates an example of a suitable operating environment 200 in which the invention may be implemented. The operating environment 200 is only one example of a suitable operating environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Other well known computing systems, environments, and/or

configurations that may be suitable for use with the invention include, but are not limited to, personal computers, server computers, held-held or laptop devices, multiprocessor systems, microprocessor-based systems, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

The invention may also be described in the general context of computer-executable instructions, such as program modules, executed by one or more computers or other devices. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Typically the functionality of the program modules may be combined or distributed in desired in various embodiments.

Network server 102 typically includes at least some form of computer readable media. Computer readable media can be any available media that can be accessed by network server 102. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, BC-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by network server 102. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other

transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

While the above embodiments of the present invention describe a network based processing system providing processing services to remote clients, one skilled in the art will recognize that the various distributed computing architectures may be used to implement the present invention as recited within the attached claims. It is to be understood that other embodiments may be utilized and operational changes may be made without departing from the scope of the present invention.

The foregoing description of the exemplary embodiments of the invention has been presented for the purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not with this detailed description, but rather by the claims appended hereto. Thus the present invention is presently embodied as a method, apparatus, computer storage medium or propagated signal containing a computer program for providing a method, apparatus, and article of manufacture for providing network based processing system providing processing services to remote clients.